RECEIVED
CENTRAL FAX CENTER
JAN 1 0 2006

WOOD, HERRON & EVANS, L.L.P.

2700 CAREW TOWER
441 VINE STREET
CINCINNATI, OHIO 45202-2917
TELEPHONE: 513-241-2324
FACSIMILE: 513-241-6234
WEBSITE:: www.whepatent.com

PATENT, TRADEMARK, COPYRIGHT
AND UNFAIR COMPETITION LAW
AND RELATED LITIGATION

EDMUND P. WOOD 1923-1968
TRUMAN A. HERRON 1935-1976
EDWARD B. EVANS 1936-1971

January 10, 2006

FACSIMILE COVER SHEET

To: Examiner Jennifer N. To
Art Unit 2195
Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22213-1450

Fax: 571-273-8300

Enclosures:
- Fax Cover Sheet containing Certificate of Facsimile Transmission (1 page)
- Appeal Brief Transmittal containing Certificate of Facsimile Transmission and Authorization to Charge Deposit Account 23-3000 in the amount of $500.00 for the Fee (2 pages)
- Appeal Brief (16 total pages, including Cover Sheet, 10 pages Appeal Brief and 5 pages Claims Appendix)

From: Douglas A. Scholer
Reg. No. 52,197

Re: U.S. Patent Application

Serial No.	10/022,982
Filed:	December 17, 2001
Applicant:	William Joseph Armstrong et al.
Art Unit:	2195
Confirmation No.:	4230
Attorney Docket:	ROC920010097US1
Our Ref:	IBM/195

Pages: 19 (including cover sheet)

MESSAGE/COMMENTS
OFFICIAL

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that this correspondence and the enclosures noted herein (19 total pages, including cover sheet) are being transmitted via facsimile transmission to Examiner Jennifer N. To, Art Unit 2195, Mail Stop Appeal Brief-Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 at 571-273-8300 on January 10, 2006.

Judith L Volk — January 10, 2006
Judith L. Volk — Date

If transmission is interrupted or of poor quality, please notify us immediately by calling (513) 241-2324 and ask for the sender's assistant. **OUR FAX NUMBER IS (513) 241-6234.**

RECEIVED
CENTRAL FAX CENTER
JAN 1 0 2006

PATENT

IBM/195
Confirmation No. 4230

CERTIFICATE OF FACSIMILE TRANSMISSION

I hereby certify that this correspondence and the enclosures noted herein 19 total pages, including cover sheet) are being transmitted via facsimile transmission to Examiner Jennifer N. To, Art Unit 2195, Mail Stop Appeal Brief-Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 at 571-273-8300 on January 10, 2006.

Judith L. Volk — January 10, 2006
Judith L. Volk Date

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	William Joseph Armstrong et al.	Art Unit:	2195
Serial No.:	10/022,982	Examiner:	Jennifer N. To
Filed:	December 17, 2001	Atty. Docket No.:	ROC920010097US1

For: DYNAMIC DIAGNOSTIC PROGRAM FOR DETERMINING THREAD WAIT TIME

Cincinnati, Ohio 45202 January 10, 2006

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF (PATENT APPLICATION-37CFR 191)

1. Transmitted herewith is the APPEAL BRIEF in this application with respect to the Notice of Appeal received by the Office on November 10, 2005.

2. **STATUS OF APPLICANT**

   This application is on behalf of

   XX **other than a small entity**

   ____ small entity

   Verified Statement:

   ____ attached

   ____ already filed

3. **FEE FOR FILING APPEAL BRIEF**

   Pursuant to 37 CFR 1.17(f) the fee for filing the Appeal Brief is:

____	Small entity	$250.00
XX	**Other than a small entity**	**$500.00**

RECEIVED
CENTRAL FAX CENTER

JAN 1 0 2006

4. **EXTENSION OF TIME**

Applicant petitions for an extension of time under 37 C.F.R. 1.136(a) for the total number of months checked below:

Months		**Fee for other than small entity**	**Fee for small entity**
_______	one month	$ 120.00	$ 60.00
_______	two months	.... 450.00	.... 225.00
_______	three months	..... 1,020.00	.... 510.00
_______	four months	...... 1,590.00	.... 795.00
_______	five months	...... 2,160.00	...... 1,080.00

Fee: $ _____

If an additional extension of time is required, please consider this a petition therefor.

5. **TOTAL FEE DUE**

The total fee due is:

Appeal brief fee $500.00
Extension fee ________

6. **FEE PAYMENT**

_____ Attached is a check in the sum of $ _______

XX Charge fee of $500.00 to Deposit Account No. 23-3000.

7. **FEE DEFICIENCY**

XX Charge any additional extension fee required or credit any overpayment to Deposit Account No. 23-3000.

WOOD, HERRON & EVANS, L.L.P.

By [illegible]
Douglas A. Scholer
Reg. No. 52,197

2700 Carew Tower
441 Vine Street
Cincinnati, Ohio 45202
(513) 241-2324

Attorney Docket No. IBM/195
Confirmation No. 4230

PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte William Joseph Armstrong, Ryan Harvey Bishop, Michael Brian Brutman, Chris Francois, Richard Karl Kirkman, Jay Paul Kurtz, Henry Joseph May, Naresh Nayer and Dennis A. Towne

Appeal No. ___________
Application No. 10/022,982

APPEAL BRIEF

RECEIVED
CENTRAL FAX CENTER

JAN 1 0 2006

PATENT

Attorney Docket No. IBM/195
Confirmation No. 4230

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	William Joseph Armstrong et al.	Art Unit:	2195
Serial No.:	10/022,982	Examiner:	Jennifer N. To
Filed:	December 17, 2001	Atty. Docket No.:	IBM/195

For: DYNAMIC DIAGNOSTIC PROGRAM FOR DETERMINING THREAD WAIT TIME

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

I. REAL PARTY IN INTEREST

This application is assigned to International Business Machines Corporation of Armonk, New York.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF CLAIMS

Claims 1-38 are pending in the Application, with claims 1-2, 19-20 and 22 each being once amended. Claim 38 has been amended twice. Claim 39 was cancelled. All pending claims stand rejected and are now on appeal.

IV. STATUS OF AMENDMENTS

An Amendment After Final was filed on January 9, 2006 amending claims 22 and 38, and cancelling claim 39 in accordance with MPEP Section 1207.

01/11/2006 KBETEMA1 00000102 233000 10022982
01 FC:1402 500.00 DA

V. SUMMARY OF CLAIMED SUBJECT MATTER

Applicants' invention is generally directed to the evaluation of program performance within a multithreaded computer system. Such evaluation is needed to locate problems in code that can cause inefficiency or overburdening of processors and other computer resources. Prior to Applicants' invention, programmers struggled to efficiently analyze code within multithreaded environments. Trace algorithms and other conventional evaluation tools, which were developed for non-multithreaded environments, were hindered by the unique nature of multithreaded systems. Unlike other computer environments, where such conventional tools are required only to list data for a single thread, the multithreaded environment causes conventional tools to produce trace data for each of the multitude of threads, burdening both system and analyst with a preclusively large list. Applicants' invention avoids the overwhelming amounts of data produced by conventional trace evaluation programs in a multithreaded environment to provide useful, focused analysis.

The invention streamlines analysis by storing thread data in logical association with a resource. For instance, an embodiment of the invention may determine a time increment indicative of how long a thread waited for a particular processor. This time increment data may be stored in logical association with the resource. For example, a common identifier may be assigned to the resource, thread data, and a hash bucket configured to store the associated relationship (correlation) between the thread diagnostic data and the resource. A programmer may then view the displayed bucket contents to digest, at a glance, which bucket and corresponding resource(s) may be prone to inordinate system blockage. In another example, analysis of the buckets may reveal a resource or thread sequence of particular interest. Such information may allow a programmer to better focus their debugging efforts on problematic system and/or programming areas (Application, page 27, lines 10-22).

In this manner, Applicants' invention provides distributed, resource oriented storage of diagnostic data that focuses debugging applications on contentious processes. Such processes may include threads competing for access to a resource upon which their execution is dependent. To facilitate evaluation, diagnostic data relating to execution of the thread may accumulate within the bucket or other data structure. Exemplary diagnostic data may include how long the thread actually waits for access to a contentious resource, and may also include program code preceding a locking occurrence. Determination of thread wait time may include subtracting a time corresponding to the occurrence of the lock from a time instance corresponding to the end of the lock. Other sources of diagnostic data may relate to program code of an invocation stack and/or pointer data (Application, pg. 15).

In any case, the hash bucket or other structure displays the diagnostic data to a user such that the data appears relative to the resource. In this manner, the data may be evaluated in the context of a contentious resource that is of particular interest to the user. Where desired, the diagnostic data may be reassigned to other buckets, which may correspond to different resources, so that greater granularity may be achieved in the context of evaluating thread wait time with respect to contentious resources (Application, pg. 21).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. Claims 1-9, 18, 21-28 and 37-38 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 4,183,083 to *Chatfield* (hereinafter "*Chatfield*") in view of U.S. Patent No. 6,202,199 to *Wygodny et al.* (hereinafter "*Wygodny*").

B. Claims 10-17, 19-20 and 29-36 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over *Chatfield* in view of *Wygodny*, and further in view of U.S. Patent No. 5,872,909 to *Wilner et al.* (hereinafter "*Wilner*").

VII. ARGUMENT

Applicant respectfully submits that the Examiner's obviousness rejections of claims 1-38 are not supported on the record, and that the rejections should be reversed. Among other failings, the primary reference, *Chatfield*, fails to disclose the claimed multithreaded environment, and *Wygodny* suggests only on the conventional trace algorithms upon which the present invention improves.

A *prima facie* showing of obviousness requires that the Examiner establish that the differences between a claimed invention and the prior art "are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art." 35 U.S.C. §103(a). Such a showing requires that all claimed features be disclosed or suggested by the prior art. Such a showing also requires objective evidence of the suggestion, teaching or motivation to combine or modify prior art references, as "[c]ombining prior art references without evidence of such a suggestion, teaching or motivation simply takes the inventor's disclosure as a blueprint for piecing together the prior art to defeat patentability – the essence of hindsight." In re Dembiczak, 50 USPQ2d 1614, 1617 (Fed. Cir. 1999)."

Applicant respectfully submits that, in the instant case, the Examiner has failed to establish a *prima facie* case of obviousness as to any of the pending claims, and as such, the rejections should be reversed. Specific discussions of the non-obviousness of each of the aforementioned groups of claims are presented hereinafter.

A. **Claims 1-9, 18, 21-28 and 37-38 were improperly rejected under 35 U.S.C. § 103(a) as being unpatentable over *Chatfield* in view of *Wygodny*.**

Independent Claim 1

This claim generally recites a method of analyzing program execution within an operating system of a multithreaded environment. The method includes accumulating diagnostic data pertaining to a thread accessing a resource, and storing the diagnostic data within a data structure at a location in the data structure correlated to the resource. The execution of a thread is predicated upon the thread's access to the resource within the multithreaded environment.

Prior to embodiments of Applicants' invention, locating a problematic task and/or resource within a multithreaded environment was a daunting and painstaking undertaking that relied on trace switch and other conventional programs. Such prior art programs generated preclusively voluminous amounts of data through which a programmer had to sift. The conventional programs are hindered by the nature of a multithreaded environment, where unlike in other computer systems, threads must share a relatively complex priority scheme that partially coordinates allocation of processing cycles as between different threads. Conventional debugging techniques consequently required tracking an amount of data that is commensurate with the exponentially high activity of the environment's multiple threads. Thus, the very efficiencies afforded by such multithreaded environments require special (trace) debugging programs, and prior to Applicants' embodiment, such programs generated data that was too overwhelming and disconnected to be useful, as described in greater detail starting on page 4 of the Application.

It is these problems (uniquely associated with a multithreaded environment) that are solved by claim 1. In particular, "diagnostic data pertaining to a thread" is analyzed "within an operating system of a multithreaded environment." In contrast, the Examiner admits on page 4 of the Final Office Action that the primary reference, *Chatfield*, fails to disclose a thread, let alone multiple threads. There is consequently neither a motivation nor a teaching in *Chatfield* to address debugging problems associated with the processes of multiple threads, as required by claim 1.

To remedy this deficiency, the Examiner asserts that it would have been obvious to combine the non-multithreaded processes of *Chatfield* with the multithreaded processes of *Wygodny*. However, *Wygodny* actually teaches away from such a combination at column 2, lines 33-36. That is, *Wygodny* acknowledges the additional complexities inherent to multithreaded environments that render useless those debugging programs (such as are disclosed in *Chatfield*) that are not designed for a multithreaded environment. There is consequently no motivation to combine *Chatfield* with *Wygodny*.

Even if such a hypothetical combination was motivated, in any case, the resultant combination would still fail to suggest accumulating diagnostic data pertaining to a thread accessing a resource, and storing the diagnostic data within a data structure at a location in the data structure correlated to the resource. *Wygodny* relies on the trace algorithms described in the background of Applicants' invention at page 5. These conventional trace algorithms (as taught by *Wygodny* at col. 2, lines 59-67) require a programmer to sift through screens of data. Notably, data in *Wygodny* is never stored in a data structure correlated to a resource. Trace algorithms (such as taught in *Wygodny*) merely list all diagnostic activity in chronological order. There is no suggestion or motivation to correlate the chronologically listed activity to a resource. The resultant inflexible, voluminous listing contrasts the processes of claim 1, where the thread data is logically associated, e.g., via a common identifier, with a CPU or other resource. Claim 1 processes instead allow a programmer to focus their analysis on a specific resource of interest, as opposed to trying to read through massive amounts of unrelated and unfocused data. That is, the programmer using the conventional trace algorithm disclosed in *Wygodny* is relegated to searching through volumes of material based only on the source code and elements ("other functions, processes" - col. 10, line 53), i.e., not a resource.

Similarly, there is no teaching or suggestion within Chatfield to store diagnostic data within a data structure correlated to a resource. The text cited by the Examiner, for instance, discloses storing merely either program or resource data, but does not teach or suggest correlating the two types of data, let along correlating a bucket storing the data to the resource. This correlation provides enormous efficiencies for programmers attempting to focus a debugging process within a multithreaded environment. Because neither cited reference discloses or suggests storing diagnostic thread data within a data structure correlated to a resource, the proposed combination falls short of establishing a prima facie case of obviousness. The rejection of independent claim 1 should therefore be reversed.

Dependent Claims 2-6 and 18

Claims 2-6 and 18 are not argued separately.

Dependent Claims 7-9

Claims 7-9 are patentable by virtue of their dependency on claim 1. However, these claims additionally include *inter alia* the feature of correlating/matching, or otherwise assigning an identifier to the resource and the data structure. The cited prior art merely lists activity, and does not suggest or motivate matching a resource to thread activity, let along matching both to an identifier. This logical association assists programmers in locating and analyzing specific thread activity and resource utilization.

Independent Claim 21

Next turning specifically to the rejection of independent claim 21, this claim generally recites an apparatus that includes at least one processor configured to execute a plurality of threads, a memory and program code resident in the memory that is configured to execute on the at least one processor. The program code is also configured to accumulate diagnostic data pertaining to a thread accessing a resource. The execution of a thread is predicated upon the thread's access to the resource. The program code is further configured to store the diagnostic data within a data structure at a location in the data structure correlated to the resource.

Applicants respectfully submit that independent claim 21 is novel and non-obvious over the prior art cited by the Examiner for reasons similar to those presented with regard to claim 1. Specifically, the combination of *Chatfield* and *Wygodny* fails to disclose or suggest the concept of storing diagnostic data within a data structure at a location in the data structure correlated with a resource, in combination with the other features recited in the claim (including a multithreaded environment). The rejection of independent claim 21 should therefore be reversed.

Dependent Claims 22-25 and 37

Claims 22-25 and 37 are not argued separately.

Dependent Claims 26-28

Claims 26-28 are patentable by virtue of their dependency on claim 21. However, these claims additionally include *inter alia* the feature of correlating/matching, or otherwise assigning an identifier to the resource and the data structure. The cited prior art merely lists activity, and does not suggest or motivate matching a resource to thread activity, let along matching both to an identifier. This logical association assists programmers in locating and analyzing specific thread activity and resource utilization.

Independent Claim 38

Claim 38 generally recites a program product that includes program code for analyzing program execution within an operating system of a multithreaded environment, wherein the program code is configured to accumulate diagnostic data pertaining to a thread accessing a resource, the execution of a thread being predicated upon the thread's access to the resource, and to store the diagnostic data within a block of the memory correlated to the resource, and a signal bearing medium bearing the program code.

Applicants respectfully submit that independent claim 38 is non-obvious over the prior art cited by the Examiner for reasons similar to those presented with regard to claim 1. Specifically, the combination of *Chatfield* and *Wygodny* fails to disclose or suggest the concept of storing diagnostic data within a data structure at a location in the data structure correlated with a resource, in combination with the other features recited in the claim. The rejection of independent claim 38 should therefore be reversed.

A. **<u>Claims 10-17, 19-20 and 29-36 were improperly rejected under 35 U.S.C. § 103(a) as being unpatentable over *Chatfield* in view of *Wygodny*, and further in view of *Wilner*.</u>**

<u>Independent Claim 19</u>

Claim 19 generally recites a method of analyzing program execution within a computer system having a plurality of threads accessing a plurality of resources. The method includes calculating a time increment reflective of a duration a thread of the plurality of threads waits for access to a resource of the plurality of resources, and storing the time increment within a bucket of a plurality of buckets comprising a hash array, each bucket being correlated to the resource. The execution of the thread is predicated upon the thread's access to the resource.

As discussed above in connection with claim 1, the combination of *Chatfield* with *Wygodny* used to reject claim 19 is unmotivated. Similarly, *Wygodny* teaches at column 2, lines 33-36 against a combination with *Wilner*, which like *Chatfield*, fails to contemplate a multithreaded environment. There is consequently no proper motivation to combine the references.

However, even a hypothetical combination of the three references would still fail to suggest storing the time increment within a bucket of a plurality of buckets comprising a hash array, each bucket being correlated to the resource. *Wygodny* and *Chatfield* do not suggest correlating data, let alone a time increment, in a bucket that is correlated to a resource. *Wygodny* merely displays program data in the manner of a conventional trace algorithm, and *Chatfield* does not correlate resources to a hash bucket. Although *Wilner* mentions "time stamps," it does not calculate a time increment reflective of a duration of a thread (nor does it even contemplate a thread). A time stamp merely records the current time of an event. There is no disclosure or suggestion within *Wilner* of subtracting or otherwise processing time stamps to determine an increment of time, e.g., corresponding to the total time a thread is locked out from a resource.

Wilner furthermore provides no remedy to the above deficiencies relating to an absence of storing the time increment within a bucket of a plurality of buckets comprising a hash array, each bucket being correlated to the resource. Because none of the cited references discloses or suggests storing a time increment within a bucket of a plurality of buckets comprising a hash array - each bucket being correlated to the resource, the proposed combination falls short of establishing a prima facie case of obviousness. The rejection of independent claim 19 should therefore be reversed.

Dependent Claim 20

Claim 20 is not argued separately.

Dependent Claims 10-17

Claims 10-17 are allowable by virtue of their dependency on Claim 1, as discussed above. Moreover, these claims each recite *inter alia* the concept of determining a time increment corresponding to a duration that a thread remains locked. As described above in connection with claim 19, no combination of the cited prior art suggests determining a time increment. For at least this reason, the reversal of the Examiner's rejections of claim 10-17 is respectfully requested.

Dependent Claims 29-36

Claims 29-36 are allowable by virtue of their dependency on Claim 21, as discussed above. Moreover, these claims each recite *inter alia* the concept of determining a time increment corresponding to a duration that a thread remains locked. As described above in connection with claim 19, no combination of the cited prior art suggests determining a time increment. For at least this reason, the reversal of the Examiner's rejections of claim 29-36 is therefore respectfully requested.

VIII. CONCLUSION

In conclusion, Applicants respectfully request that the Board reverse the Examiner's rejections of claims 1-38, and that the Application be passed to issue. If there are any questions regarding the foregoing, please contact the undersigned at 513/241-2324. Moreover, if any other charges or credits are necessary to complete this communication, please apply them to Deposit Account 23-3000.

Respectfully submitted,

WOOD, HERRON & EVANS, L.L.P.

Date: 1/10/06

By: [illegible]
Douglas A. Schoder
Reg. No. 52,197

2700 Carew Tower
441 Vine Street
Cincinnati, Ohio 45202
(513) 241-2324 - Telephone
(513) 241-6234 - Facsimile

IX. CLAIMS APPENDIX: CLAIMS ON APPEAL (S/N 10/022,982)

1. (Once Amended) A method of analyzing program execution within an operating system of a multithreaded environment, comprising:

accumulating diagnostic data pertaining to a thread accessing a resource, the execution of a thread being predicated upon the thread's access to the resource within the multithreaded environment; and

storing the diagnostic data within a data structure at a location in the data structure correlated to the resource.

2. (Once Amended) The method according to claim 1, wherein the diagnostic data includes data selected from at least one of: a time measurement, program code executed by the thread, an invocation stack, and pointer data.

3. (Original) The method according to claim 1, wherein the data structure comprises a hash bucket.

4. (Original) The method according to claim 1, further comprising determining the resource.

5. (Original) The method according to claim 4, wherein determining the resource includes reading contents of a task dispatcher.

6. (Original) The method according to claim 1, further comprising storing information identifying the resource.

7. (Original) The method according to claim 1, further comprising matching an identifier corresponding to the resource to a correlative identifier corresponding to the data structure.

8. (Original) The method according to claim 7, further comprising reassigning the identifier to a second resource.

9. (Original) The method according to claim 7, further comprising assigning the correlative identifier to the data structure.

10. (Original) The method according to claim 1, further comprising detecting a locking occurrence.

11. (Original) The method according to claim 10, further comprising calculating a time increment corresponding to a duration that the thread remains locked.

12. (Original) The method according to claim 11, further comprising storing the time increment within the data structure.

13. (Original) The method according to claim 10, further comprising recording the time corresponding to the locking occurrence.

14. (Original) The method according to claim 1, further comprising detecting a removal of the lock.

15. (Original) The method according to claim 14, further comprising recording a time instance corresponding to the removal of the lock.

16. (Original) The method according to claim 10, further comprising recording program data relating to code executed by the thread prior to the locking occurrence.

17. (Original) The method according to claim 16, further comprising retrieving the program data from an invocation stack.

18. (Original) The method according to claim 1, further comprising displaying the diagnostic data.

19. (Once Amended) A method of analyzing program execution within a computer system having a plurality of threads accessing a plurality of resources, comprising:

calculating a time increment reflective of a duration a thread of the plurality of threads waits for access to a resource of the plurality of resources, the execution of the thread being predicated upon the thread's access to the resource; and

storing the time increment within a hash bucket of a plurality of hash buckets comprising a hash array, each hash bucket being correlated to the resource.

20. (Once Amended) The method according to claim 19, further comprising reallocating the plurality of resources to the plurality of hash buckets to group the diagnostic data with a different scheme.

21. (Original) An apparatus comprising:

at least one processor configured to execute a plurality of threads;

a memory; and

program code resident in the memory and configured to execute on the at least one processor, the program code configured to accumulate diagnostic data pertaining to a thread accessing a resource, the execution of a thread being predicated upon the thread's access to the resource, and to store the diagnostic data within a data structure at a location in the data structure correlated to the resource.

22. (Once Amended) The apparatus according to claim 21, wherein the diagnostic data includes data selected from a group consisting of at least one of: a time measurement, program code executed by the thread, an invocation stack and pointer data.

23. (Original) The apparatus according to claim 21, wherein the lock of memory comprises a hash bucket.

24. (Original) The apparatus according to claim 21, wherein the program code initiates a determination of the resource.

25. (Original) The apparatus according to claim 21, wherein the program code initiates storing information identifying the resource.

26. (Original) The apparatus according to claim 21, further comprising matching an identifier corresponding to the resource to a correlative identifier corresponding to the data structure.

27. (Original) The apparatus according to claim 26, wherein the program code initiates reassigning the identifier to a second resource.

28. (Original) The apparatus according to claim 26, wherein the program code initiates assigning the correlative identifier to the data structure.

29. (Original) The apparatus according to claim 21, wherein the program code initiates a detection of a locking occurrence.

30. (Original) The apparatus according to claim 21, wherein the program code initiates a calculation of a time increment corresponding to a duration that the thread remains locked.

31. (Original) The apparatus according to claim 30, wherein the program code initiates storing the time increment within the data structure.

32. (Original) The apparatus according to claim 21, wherein the program code initiates recording a time corresponding to a locking occurrence.

33. (Original) The apparatus according to claim 21, wherein the program code initiates detecting a removal of the lock.

34. (Original) The apparatus according to claim 33, wherein the program code initiates recording a time instance corresponding to the removal of the lock.

35. (Original) The apparatus according to claim 29, wherein the program code initiates recording program data relating to code executed by the thread prior to a locking occurrence.

36. (Original) The apparatus according to claim 35, wherein the program code initiates retrieval of the program data from an invocation stack.

37. (Original) The apparatus according to claim 21, wherein the program code initiates a display of the diagnostic data.

38. (Twice Amended) A program product, comprising:

program code executable by a computer for analyzing program execution within an operating system of a multithreaded environment, wherein the program code is configured to accumulate diagnostic data pertaining to a thread accessing a resource, the execution of a thread being predicated upon the thread's access to the resource, and to store the diagnostic data within a block of the memory correlated to the resource; and

a recordable medium bearing the program code.

39. (Cancelled)